Translation-based Supervision for Policy Generation in Simultaneous Neural Machine Translation

Ashkan Alinejad, Hassan S. Shavarani and **Anoop Sarkar**
School of Computing Science
Simon Fraser University
BC, Canada
{aalineja,sshavara,anoop}@sfu.ca

Abstract

In simultaneous machine translation, finding an agent with the optimal action sequence of reads and writes that maintain a high level of translation quality while minimizing the average lag in producing target tokens remains an extremely challenging problem. We propose a novel supervised learning approach for training an agent that can detect the minimum number of reads required for generating each target token by comparing simultaneous translations against full-sentence translations during training to generate oracle action sequences. These oracle sequences can then be used to train a supervised model for action generation at inference time. Our approach provides an alternative to current heuristic methods in simultaneous translation by introducing a new training objective, which is easier to train than previous attempts at training the agent using reinforcement learning techniques for this task. Our experimental results show that our novel training method for action generation produces much higher quality translations while minimizing the average lag in simultaneous translation.

1 Introduction

Simultaneous Machine Translation focuses on the real-time translation of the stream of utterances in the source language to the target language. The essence of simultaneous translation imposes a trade-off between translation quality and the delay in the delivery of the translated utterances. Finding the optimal segments in the input stream is one important task to balance the delay and the translation quality. Segmentation based on the input sentence structure (Ryu et al., 2006; Oda et al., 2014; Shavarani et al., 2015) has been explored in previous work.

The other approach to reach the optimal segmentation strategy (policy) is to define the segmentation problem in reinforcement learning framework (Satija and Pineau, 2016; Gu et al., 2017; Alinejad et al., 2018) in which the segmentation agent chooses among two possible actions of READ (waiting to receive more input) and WRITE (producing an output based on the available input inventory).

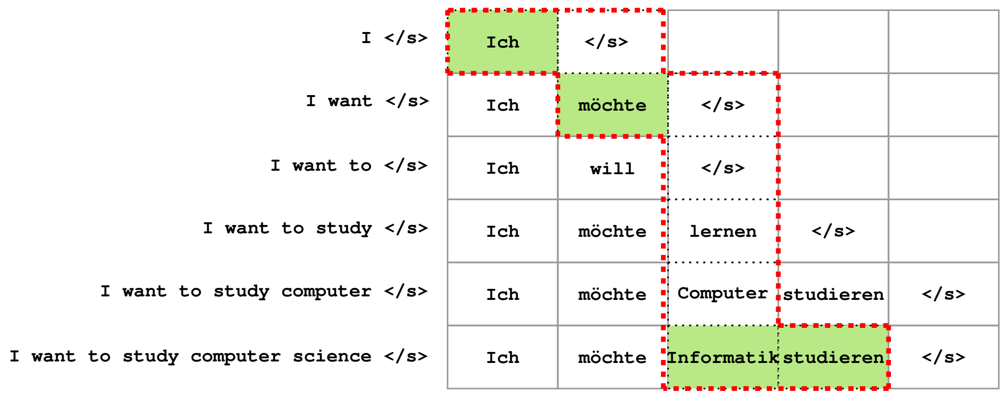
The segmentation agents can either use a fixed-latency policy (Dalvi et al., 2018; Ma et al., 2019a) or use an adaptive policy (Grissom II et al., 2014). In the former, the agent waits for receiving a certain number of input tokens and performs pairs of one WRITE and one READ afterward. In the latter, the agent looks for a specific signal in input to permit a WRITE action; a signal that might come from a parsing feature (Grissom II et al., 2014), a stochastic classifier (Gu et al., 2017), or the attention module of the translation model (Arivazhagan et al., 2019; Ma et al., 2020).

Recently, imitation learning has been considered to train adaptive policies. This thread of research focuses on designing supervised oracle agents that can compute the optimal sequence of READ/WRITE actions; a sequence that leads the model to produce the most similar translation to that of an offline translation model[1]. Zheng et al. (2019a,b) and Arthur et al. (2021) use pairs of (input, reference) sentences to train the oracle agent.

In this work, we create such an oracle agent using a fully trained neural machine translation model. We do not use reference translations to improve our agent and the translation model is not fine-tuned based on the performance of the agent. In spite of that, we are successful at lowering the latency of simultaneous translation below all previous methods across many different language pairs while retaining a competitive translation quality. Our results also demonstrate the applicability of our policy in augmenting existing simultaneous neural translation approaches and improving their

[1]A model that waits to receive all the input stream tokens and then starts to translate. Such a model can provide more accurate translations in comparison to simultaneous translation model that performs the task using partial and incomplete information.

(a) Example of a created Partial Translations Table



I </s>	🟩 Ich	</s>			
I want </s>	Ich	🟩 möchte	</s>		
I want to </s>	Ich	will	</s>		
I want to study </s>	Ich	möchte	lernen	</s>	
I want to study computer </s>	Ich	möchte	Computer	studieren	</s>
I want to study computer science </s>	Ich	möchte	🟩 Informatik	🟩 studieren	</s>

(b) The created optimal segments in the input stream based on the Partial Translations Table above. The reference action sequence of READs ($\mathfrak{R}$) and WRITEs ($\mathfrak{W}$) can be created based on the occupied cells in each column.

$\mathfrak{R}$	I		want		to	study	computer	science		
$\mathfrak{W}$		Ich		möchte					Informatik	studieren

Figure 1: Example of partial translations table. Our oracle action trajectory is indicated by the red dash line and the green cells are the words that our policy choose to WRITE. The subset of input words at each row is extended with </s> token to improve the quality of partial translations.

translation quality and latency.

The remaining parts of the paper are as follows. In Section 2, we formally define our imitation learning problem and describe our solution to it. In Section 3, we examine our provided solution, and in Section 4, we examine the solution and provide experimental results and the analysis of our results. Section 5 compares our work to the related work and Section 6 concludes our work.

2 Supervised Approach

In simultaneous machine translation, we aim to receive the input sequence $X = \{x_1, \dots, x_J\}$ incrementally and to transform it to the translated tokens $Y = \{y_1, \dots, y_I\}$ as accurately and as fast as possible (by producing the output while reading the input). Our supervised framework contains two main components: The INTERPRETER which takes subsets of the input sequence $X_j = \{x_1, \dots, x_j\}$ and generates *partial* translations $Y^j = \{y^j_1, \dots, y^j_{m_j}\}$[2]; And the AGENT which decides whether to send the next input subset X_{j+1} to the INTERPRETER or not, based on the currently generated output tokens (and possibly other useful information) which represents the state of the INTERPRETER.

[2]The last partial translation equals to the full-sentence translation. i.e. $Y^J = Y$

2.1 Reference Action Sequences

The central idea behind our method is our novel definition of an *optimal segment* in simultaneous translation. We define an *optimal segment* as a segment of the input sequence which leads the INTERPRETER to produce the exact same target words in both simultaneous (partial) translation and full-sentence translation. The initial optimal segment is a prefix of the input and each subsequent optimal segment is a further slice of the input. A reference action sequence (or an oracle action sequence) is a sequence that splits the input sentence into optimal segments.

To achieve this goal we build a *Partial Translations Table* (PTT), each row of which corresponds to the translation of a prefix of the source sentence and each column represents a translated word. More explicitly, the first row contains the translation of the first input token followed by the end-of-sentence token </s> and each next row will incrementally consider one more input token than the immediate row before. By definition, the last row will be equivalent to the full-sentence translation. Figure 1(a) shows an example of a generated Partial Translations Table.

We construct the reference action sequence of optimal segments using the PTT. Starting from the leftmost word in the first row, we compare the content of each column with the word in its own column at the last row. If they are the same we add

WRITE action to the oracle sequence and move to the right cell. Otherwise, we will add READ and move down to the lower cell.

Figure 1(b) demonstrates the optimal segments created using PTT for the example input sentence "`I want to study computer science`". Using this stream, the first element of the action sequence will be $\mathfrak{R}$ since the READ row is occupied in the first column. The next element will be $\mathfrak{W}$ since the WRITE row is occupied in the second column. If we continue to look at each column and generate a READ or WRITE action we will end up with the reference action sequence of "$\mathfrak{R}\,\mathfrak{W}\,\mathfrak{R}\,\mathfrak{W}\,\mathfrak{R}\,\mathfrak{R}\,\mathfrak{R}\,\mathfrak{R}\,\mathfrak{W}\,\mathfrak{W}$".

We now formally define the reference action sequence generation algorithm. Let j be the number of READs and i be the number of WRITEs at a given time step $t = i + j$. At time step $t + 1$, if the token y_i in the full-sentence translation Y, equals to the i'th token in the partial translation Y^j, it means that the current subset of the input words X_j is sufficient to generate the i'th word in the output and our agent should choose to WRITE. Otherwise, we will add a READ to our action sequence. Algorithm 1 defines the extraction process.

Our proposed algorithm for generating the reference action sequence is completely agnostic about the underlying partial translation generation component. We can choose offline (non-simultaneous) or simultaneous translation models to create the Partial Translations Table.

Algorithm 1 Generating action sequences

```
1:  Init $i \leftarrow 1$, $j \leftarrow 1$, actions $\leftarrow$ [R]
2:  $Y$ = Translate($X_J$)
3:  while $i <$ len($Y$) do
4:      $Y^j$ = Translate($X_j$)
5:      if $i <$ len($Y^j$) and $Y^j_i = Y_i$ then
6:          actions $\leftarrow$ actions +[W]
7:          $i \leftarrow i + 1$
8:      else
9:          actions $\leftarrow$ actions +[R]
10:         if $j < J$ then
11:             $j \leftarrow j + 1$
12: return actions
```

2.2 Supervised Training

For any input sentence X and its corresponding partial translations $Y^1, \ldots, Y^J$, we can generate an oracle action sequence $A = \{a_1, \ldots, a_T\}$, where $T = I + J$. This action sequence will be used as the ground-truth for training our action generation policy in a supervised framework. At time step t, the policy observes the current state of the INTERPRETER o_t by receiving x_i and y_j alongside a history of actions η_h from previous time steps, where $\eta_h = \{a_{t-1}, \ldots, a_{t-h}\}$. We train a recurrent neural network (RNN) to maximize the probability of the current action a_t given all the previous observations and actions:

$$\max P(a_t | a_{<t}, o_{\leq t}; \theta)$$

where θ is the set of parameters for our RNN.

At each time step, we pass the source token, the target token, and all the actions in η_h through separate embedding layers. The action embedding layer is shared among actions. The source and target embeddings will be concatenated and go through a linear layer. A separate linear layer receives concatenated action embeddings to extract features. The concatenation of the outputs of the two linear transformation layers go through 4 LSTM layers to predict the next action. Figure 2 depicts the structure of our Agent. While the agent is modeled as an LSTM, the underlying translation system we use is a Transformer NMT model.

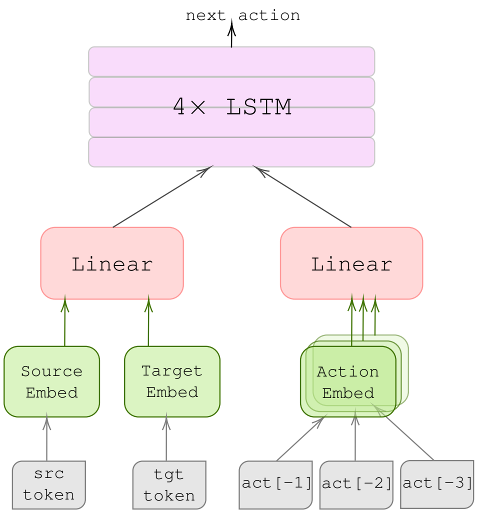


Figure 2: Architecture of our Agent. Passing the last 3 generated actions alongside the source and target tokens help the model prevent long consecutive sequences of READs or WRITEs.

2.3 Improving Robustness

Minimizing the discrepancy between training and inference, also known as exposure bias (Ranzato et al., 2016), is an essential step in successfully training a supervised agent. Our Agent has to learn how to generalize when facing unseen partial translations and make sure the errors do not compound with each mistake in its trajectory.

The Interpreter Since we do not change the translation component, if we consider the previously generated tokens instead of the ground truth translations, we can guarantee that training and inference are using the same procedure in creating the Partial Translations Table. Although this makes the training process slower, it provides more accurate results.

The Agent To prevent landing in unfamiliar areas of the prediction space with certain Agent mistakes we augment our training data of action sequences with additional examples that are introduce distortions in the action sequence (Arthur et al., 2021). For each input sentence X we generate its oracle action sequence A and the set of observations $O = \{o_1, \ldots, o_T\}$ with $o_t = (x_i, y_j, \eta_h)$ to be used for training the Agent. Then we randomly choose a time step t and check the training example (o_t, a_t) to see if it has the following conditions:

- $t \notin \{1, T\}$.
- $x_i \neq$ `</s>`.
- $y_j \neq$ `</s>`.

If all of the conditions are true, then we swap the action a_t from READ to WRITE or vice versa, and we generate a new oracle action sequence for the rest of the sentence. The observation o_t is updated according to the newly generated oracle. An example is presented and discussed in the appendix.

Beam search vs. Greedy decoding Following previous work, to boost the simultaneous nature of the model, we use greedy decoding while performing the simultaneous decoding in the INTERPRETER.

The simultaneous decoder (aka the INTERPRETER) can use beam search to get more accurate results at expense of the translation speed. However, this modification does not substantially change the comparison with baseline methods, so we leave this for future work.

Since the Partial Translation Table can be generated in an offline manner (and can be considered a pre-processing step), we use beam search decoding when creating these partial translations without any negative effect on inference for simultaneous translation at decoding time.

3 Experimental Setup

3.1 Dataset

We use IWSLT14 (Cettolo et al., 2014) and WMT15[3] German to English and IWSLT15 (Luong et al., 2015) Vietnamese to English translation tasks to examine the effectiveness of our approach.

Following Elbayad et al. (2020),we tokenize and lower-case the German to English data and BPE (Sennrich et al., 2016) sub-word tokenize both sides.

For IWSLT14 data, we choose 10K separate BPE merge operations resulting in approximately 8.8K German and 6.6K English sub-word types. We train our models on 160K sentences and keep 7K of the train data as the validation set. We test our models on a concatenation of dev2010 and tst2010 to tst2013 (a total of 6750 sentence pairs).

For WMT15, we choose BPE merge operations such that we achieve a joint BPE vocabulary of size 32K types. We will randomly choose 20 percent of the sentence pairs for training the Agent[4]. We will also use the same subset for distorting the samples and together we will end up having 1.5M training examples. We use newstest2013 with 3000 sentence pairs as the validation set and test on newstest2015 with 2169 sentences.

For IWSLT15 Vietnamese to English, we use the tokenized corpus prepared by Luong et al. (2015) which contains 17K English and 7.7K Vietnamese types. The data contains 133K training sentence pairs. We use tst2012 (1553 sentence pairs) for validation and tst2013 (1268 sentence pairs) to test our models.

3.2 Evaluation

We evaluate the translation quality of the translated sentences using tokenized word-level BLEU score (Papineni et al., 2002)[5]. We use Average Lagging (AL) (Ma et al., 2019b) to measure the decoding latency for our models. AL measures the average number of words we are lagging behind a policy that produces words at a rate proportional to the ratio between target and source lengths, with no delay.

Our goal in this paper is to minimize the decoding latency (get the lowest average lagging possi-

[3] `http://www.statmt.org/wmt15/`

[4] Making use of more data did not affect our results.

[5] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl`

	DE→EN		VI→EN	
	BLEU	AL	BLEU	AL
wait-∞ + eval-wait-5	27.92	4.48	19.93	**3.5**
wait-∞ + oracle policy	**34.25**	**4.14**	**28.29**	4.6
multi-path + eval-wait-5	30.50	4.67	20.18	**2.33**
multi-path + eval-wait-5 + oracle policy	**33.04**	**3.75**	**25.25**	4.35
wait-∞ + (Zheng et al., 2019a)	31.18	4.50	20.71	3.31

Table 1: Comparison between different oracles on IWSLT14 DE → EN and IWSLT15 VI → EN datasets.

ble) while trying to balance the loss in the translation quality as measured by BLEU score. Among the two measures, BLEU normally gets lower in production settings, as the data is not as clean and well prepared as the benchmark data. On the other hand, improving average lagging is a more reliable method to improve the user experience (in simultaneous translation).

3.3 Model Configuration

We use fairly standard Transformer-based NMT system as implemented in Fairseq (Ott et al., 2019) for all of our experiments[6]. We augment the implementation to perform simultaneous translation and we incorporate our Agent trained to produce read and write actions with the base NMT system and decoder. Our INTERPRETER can be additionally configured to replicate the model proposed in (Elbayad et al., 2020).

Our agent consists of 4 unidirectional LSTM layers (Hochreiter and Schmidhuber, 1997) with 512 units in each layer. We use a history of the last 3 previous action tokens (i.e. $h = 3$). Each embedding and linear layer generates a vector of dimension 512.

We train our Agent using Adam (Kingma and Ba, 2014) optimizer. The initial learning rate is set to 0.0008 and we use a fixed learning rate scheduler with a shrink factor of 0.95.

After training for 50 epochs, our agent that is trained to produce optimal segments obtains an average accuracy of 92.3% on the training data and 83% on the dev set when averaged over 4 experiments on the IWSLT datasets.

[6]The implementation is available at: https://github.com/sfu-natlang/Supervised-Simultaneous-MT.

4 Results and Analysis

We compare the performance of our system against two baselines:

- **Wait-∞** also known as Wait-Until-End, where the full sentences are read during training before generating any translations . This is an extreme case of *Wait-k* strategy (Ma et al., 2019a) in which the model reads the first k words of the input and then performs consecutive WRITE/READ actions afterwards.
- **Multi-path** model proposed by (Elbayad et al., 2020). The multi-path model jointly trains the translation component on decoding strategies with various latencies, which makes this model effective on a wide range of delay values.

For both of these settings, we will use the *eval-wait-k* (Ma et al., 2019a) policy, where each written word is exactly k words behind the source side. We compare the performance of our model against the state-of-the-art (SotA) in section 4.2.

4.1 Performance of Oracle Policy

Table 1 compares the performance of our policy for DE → EN and VI → EN language pairs on IWSLT dataset. *wait-∞ + eval-wait-5* corresponds to the model in which we decode an offline translation component using wait-5 policy. In *wait-∞ + oracle policy*, we use an offline INTERPRETER to generate partial translations in PTT, and then we use our algorithmic oracle to generate policies. The policy in *multi-path + eval-wait-5* is generated by decoding multi-path model with wait-5 policy. *multi-path + eval-wait-5 + oracle policy* is the model where we first use the multi-path model with wait-5 decoding path to generate partial translation and then we use our algorithm to generate oracle policy. Our choice of eval-wait-5 for decoding the multi-path model

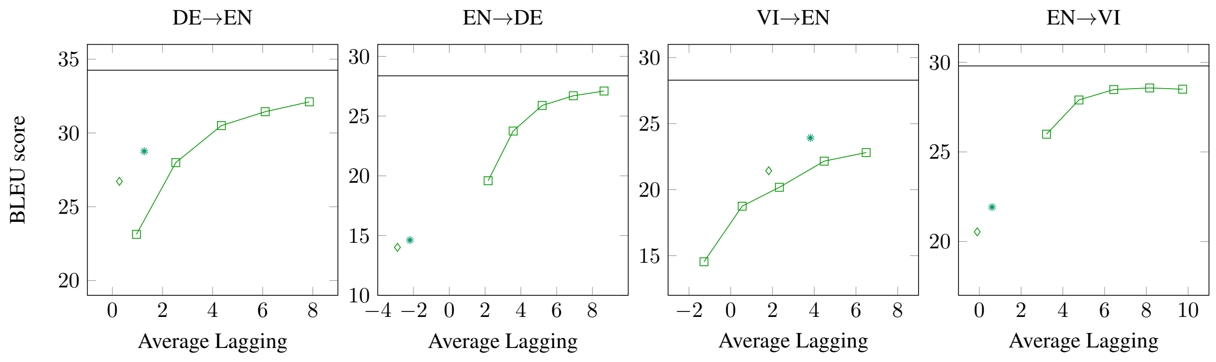


Figure 3: Translation quality vs delay of our oracle policy compared to multi-path policy. Markers: —— represents wait-∞ model. —□— corresponds to the multi-path model. each point in the curve is generated using eval-wait-k policy for k $\in \{1, 3, 5, 7, 9\}$. ∗ points to the wait-∞ + our trained policy. ◊ represents the multi-path + eval-wait-5 + our trained policy.

is based on the fact that the multi-path model generates more accurate translations via eval-wait-5 compared to offline decoding (Elbayad et al., 2020). The numbers in the last row of Table 1 are generated by using an offline INTERPRETER and the oracle in (Zheng et al., 2019a).

4.1.1 Our oracle policy vs. static policies

On both offline and multi-path settings, our oracle policy outperforms *eval-wait-5* policy both in terms of translation quality and latency on DE $\rightarrow$ EN language direction. Our policy on VI $\rightarrow$ EN experiment is slightly more delayed, but the translation quality is considerably more accurate. The performance of the *eval-wait-5* policy improves when we replace the offline INTERPRETER with the multi-path model. However, the multi-path model does not outperform *eval-wait-5* combined with our oracle policy.

The delay of our oracle policy decreases when we change the underlying translation component to a multi-path translation model so using a base translation model trained to handle shorter segments can be combined with an agent trained on our reference actions to improve the average lagging.

4.1.2 Our oracle policy vs dynamic policies

Zheng et al. (2019a,b); Arthur et al. (2021) propose algorithmic methods for generating oracle action sequences. The oracle in (Zheng et al., 2019b) introduces a new delay token in the target vocabulary which makes their INTERPRETER incompatible to our agent.

Arthur et al. (2021) use alignment-based segments to jointly train their policy and translation components. Only using alignments extracted from fast-align (Dyer et al., 2013) on an offline translation component gives us very low BLEU scores.

The oracle in (Zheng et al., 2019a) is the closest model to our work. Unlike our policy, they compare each word in partial translations to the *target words* to find the optimal action sequence. The numbers in Table 1 correspond to the closest results we could get by searching for the optimal hyperparameters. Our oracle policy outperforms their oracle policy in both language pairs.

4.2 Trained Agent Performance

Figure 3 shows the results of our trained policy in comparison with the policy trained with multi-path method on DE $\leftrightarrow$ EN and VI $\leftrightarrow$ EN language pairs. We will apply our policy on two different settings: (1) When we use a translation model trained on full sentences to fill up the partial translations table (offline + our policy) and (2) generating translations in PTT via multi-path model decoded with eval-wait-5 trajectory (multi-path + our policy). In both settings, we are using a beam of size 5 for generating each partial translation.

First, we investigate how the capabilities of the INTERPRETER can affect the quality of the generated policy. By comparing the offline INTERPRETER (marked with star) and multi-path INTERPRETER (marked with diamond) we can see that the offline model can achieve a higher translation quality with a more delayed policy. This is in align with our experimental results with their oracles (Section 4.1) where using multi-path INTERPRETER gave us less delayed policy by sacrificing translation quality.

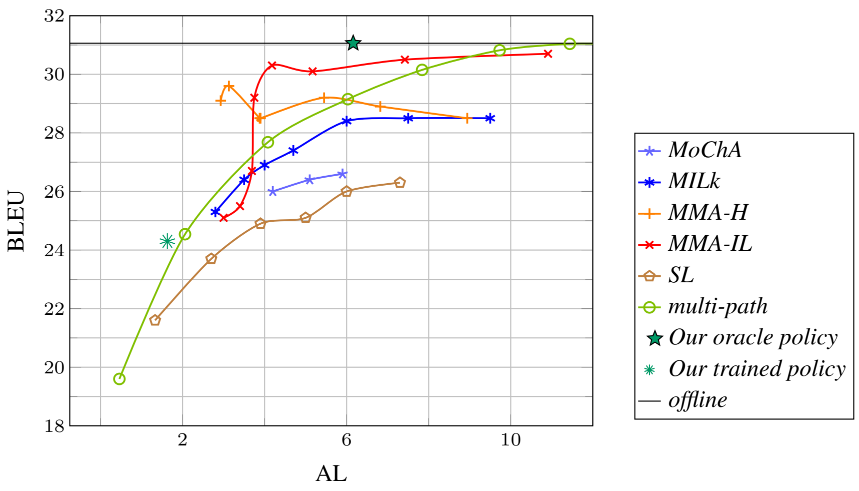


Figure 4: Comparison against SoTA results on WMT15 DE $\rightarrow$ EN language pair. SL is the supervised learning approach proposed by (Zheng et al., 2019a). "*Our oracle policy*" generates the oracle action sequences for partial translations generated by wait-∞ translation model, and "*Our trained policy*" is our Agent trained to learn that oracle.

By comparing the multi-path model (marked with square) with our trained policy we can see that in both DE $\rightarrow$ EN and VI $\rightarrow$ EN language pairs our policy outperforms the multi-path model in both settings. This is because our policy gives the INTERPRETER the freedom to translate quickly at the beginning and have consecutive reads later in the sentence, which consequently results in higher translation quality and lower latency.

In EN $\rightarrow$ DE and EN $\rightarrow$ VI our agent generates translation with much lower delays with slightly lower translation scores. This happens because translation in this direction can be monotonic without losing much in terms of translation scores. In this scenario, our model has a higher chance of making mistakes as the length of the sentence gets longer; while the multi-path model following the static policy of *eval-wait-k* obtains slightly higher translation scores.

On multi-path settings, for DE $\rightarrow$ EN language pair, our trained policy has a much lower latency while at the same time it is considerably more accurate in terms of translation scores compared to the *eval-wait-1* policy. Similarly, for VI $\rightarrow$ EN experiment, the translation quality of our policy is close to *eval-wait-7* with a delay less than *eval-wait-5* policy. This implies that we can boost the performance of the previously proposed methods by combining their translation system with an agent trained using our proposed oracle policy.

4.3 Performance on WMT15 Dataset

In order to compare the performance of our trained agent with other state-of-the-art methods, we will conduct experiments on the WMT15 DE$\rightarrow$ EN dataset. As depicted in Figure 4, our oracle policy is able to generate action sequences with AL of around 6, while our translation quality is as good as the offline model. On datasets with longer sentences like WMT, achieving oracle-level accuracy is a harder task. However, compared to a static policy like *eval-wait-k*, our Agent generates action sequences that perform similar to *eval-wait-3*. Although the policies in MoChA (Chiu and Raffel, 2018), MILK (Arivazhagan et al., 2019) and their recent version of MMA-H and MMA-IL (Ma et al., 2020) generate more accurate translations, the delay of their systems is considerably higher than our approach.

The BLEU scores of the previously proposed supervised learning approach in (Zheng et al., 2019a) is much worse than our model when we consider AL values that are similar to ours. We obtain +2 points higher BLEU score for translation quality with almost the same delay as their model.

4.4 Qualitative Analysis

Waiting for long consecutive words in the source is not always a good strategy in translating from SOV to SVO languages. Table 3 shows an example where our policy can generate each word as soon as it receives them. Such examples explain why our model performs better than *wait-k* models which

(a) Generated actions and translations via our supervised agent

ℜ	wir		mussten		uns	wegen	der	[an]	[wäl]	te	[us]	w.	sorgen	machen	.									
𝔚		we		had												to	worry	about	[law]	yers	and	so	on	.

(b) Generated actions and translations via Multi-path agent

ℜ	wir	mussten	uns	wegen	der		[an]		[wäl]		te		[us]		w.		sorgen		machen		.		
𝔚						we		had		to		look		forward		because		of		the		[law]	yers ...

Table 2: The performance comparison of generated actions from our model (a) and multi-path model (b) on the generated and reference translations. Each column shows a single READ (ℜ) or WRITE (𝔚). Subword tokens ending in @@ are shown inside brackets.

ℜ	aber		wo		sind		wir		nützlich		?	
𝔚		but		where		are		we		useful		?

Table 3: An example translation from our model where word-by-word translation generates good translations and waiting for 5 words is redundant.

are forced to wait for k words even when there is sufficient information to start the translation.

Table 2 compares the output of our model for an example German sentence to that of the multi-path model decoded with eval-wait-5 policy. As we mentioned earlier, our model does not wait too long to translate the tokens for which it has enough information. Please compare the translation time of "`wir`" and "`mussten`" in Table 2.(a) and Table 2.(b). It is apparent that eval-wait-5 must wait for 5 tokens to translate the tokens that it has already had enough information about them for a while.

In addition, our model does not translate tokens based on immature information. For example, in table 2, our model waits until it receives "`sorgen machen`" which is essential for translating "`to worry`" as opposed to the eval-wait-5 design which forces the model to produce the generic verb "`to look forward`" which causes the model to lose information about the verb. While forcing this step lowers the latency, the translated sentence becomes highly inaccurate.

5 Related Work

Satija and Pineau (2016); Gu et al. (2017) propose to use a reinforcement learning algorithm to train the Agent. Their proposed agent observes the pretrained offline translation model to learn when to READ or WRITE. Alinejad et al. (2018) improve their model by introducing a new *predict* token in order to have a better estimate of the proper action for the next time step.

Monotonic attention mechanism (Raffel et al., 2017; Chiu and Raffel, 2018) proposes to use attention mechanism to chunk the source and target sequences. However, they restrict the scope of their attention to the immediate input, which can be problematic for reordering words in the translation task. Arivazhagan et al. (2019); Ma et al. (2020) address this problem by attending to all the already seen words.

The static eval-wait-k policy proposed by Ma et al. (2019a) reads k words and then consecutively reads a word and decodes one word until the `</s>` is written to the output. The Wait-k model trains a translation component using the eval-wait-k decoding strategy which leads to generating more effective policies. Elbayad et al. (2020) show by training the translation component jointly on various wait-k trajectories, the model can generalize better over various eval-wait-k policies.

Arthur et al. (2021) propose to use word-alignments to be used as the reference for training their Agent jointly with the translation component, using imitation learning.

A similar supervised approach proposed by Zheng et al. (2019a), where an oracle is generated for a fixed INTERPRETER and then an Agent will be trained to learn it. Their approach is different from ours in that: (a) our model compares partial translations with full-sentence translations of the same INTERPRETER, which leads to having an oracle that finds notably more effective policies without any unnecessary hyper-parameters. (2) we are using a completely different architecture (3) we explore the performance of our oracle using translation components other than offline translation models.

6 Conclusion

We present a novel idea for generating optimal segments in simultaneous translation by comparing the output of a simultaneous system for translation with an offline translation model. This provides us with oracle action sequences that we can use to train an Agent used to produce read and write actions for simultaneous translation. Our experimental results show that by using an offline translation component, our agent can generate better policies in terms of translation quality and delay compared to our baselines. We also show that our agent can be trained by previously proposed translation components and generate better policies compared to what they have reported before.

References

Ashkan Alinejad, Maryam Siahbani, and Anoop Sarkar. 2018. Prediction improves simultaneous neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3022–3027, Brussels, Belgium. Association for Computational Linguistics.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, Chung-Cheng Chiu, Semih Yavuz, Ruoming Pang, Wei Li, and Colin Raffel. 2019. Monotonic infinite lookback attention for simultaneous machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1313–1323, Florence, Italy. Association for Computational Linguistics.

Philip Arthur, Trevor Cohn, and Gholamreza Haffari. 2021. Learning coupled policies for simultaneous machine translation using imitation learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2709–2719, Online. Association for Computational Linguistics.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, volume 57.

Chung-Cheng Chiu and Colin Raffel. 2018. Monotonic chunkwise attention. In *International Conference on Learning Representations*.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, and Stephan Vogel. 2018. Incremental decoding and training methods for simultaneous translation in neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 493–499, New Orleans, Louisiana. Association for Computational Linguistics.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia. Association for Computational Linguistics.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2020. Efficient Wait-k Models for Simultaneous Machine Translation. In *Interspeech 2020 - Conference of the International Speech Communication Association*, pages 1461–1465, Shangai (Virtual Conf), China.

Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. 2014. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Empirical Methods in Natural Language Processing*.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. 2017. Learning to translate in real-time with neural machine translation. In *15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Valencia, Spain.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Minh-Thang Luong, Christopher D Manning, et al. 2015. Stanford neural machine translation systems for spoken language domains. In *Proceedings of the international workshop on spoken language translation*, pages 76–79, Da Nang, Vietnam.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. 2019a. STACL: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3025–3036, Florence, Italy. Association for Computational Linguistics.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. 2019b. STACL: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. pages 3025–3036.

Xutai Ma, Juan Miguel Pino, James Cross, Liezl Puzon, and Jiatao Gu. 2020. Monotonic multihead attention. In *International Conference on Learning Representations*.

Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2014. Optimizing segmentation strategies for simultaneous speech translation. In *Proceedings of the 52nd Annual Meeting of* *the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 551–556, Baltimore, Maryland. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Colin Raffel, Minh-Thang Luong, Peter J. Liu, Ron J. Weiss, and Douglas Eck. 2017. Online and linear-time attention by enforcing monotonic alignments. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2837–2846. PMLR.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Koichiro Ryu, Shigeki Matsubara, and Yasuyoshi Inagaki. 2006. Simultaneous English-Japanese spoken language translation based on incremental dependency parsing and transfer. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 683–690, Sydney, Australia. Association for Computational Linguistics.

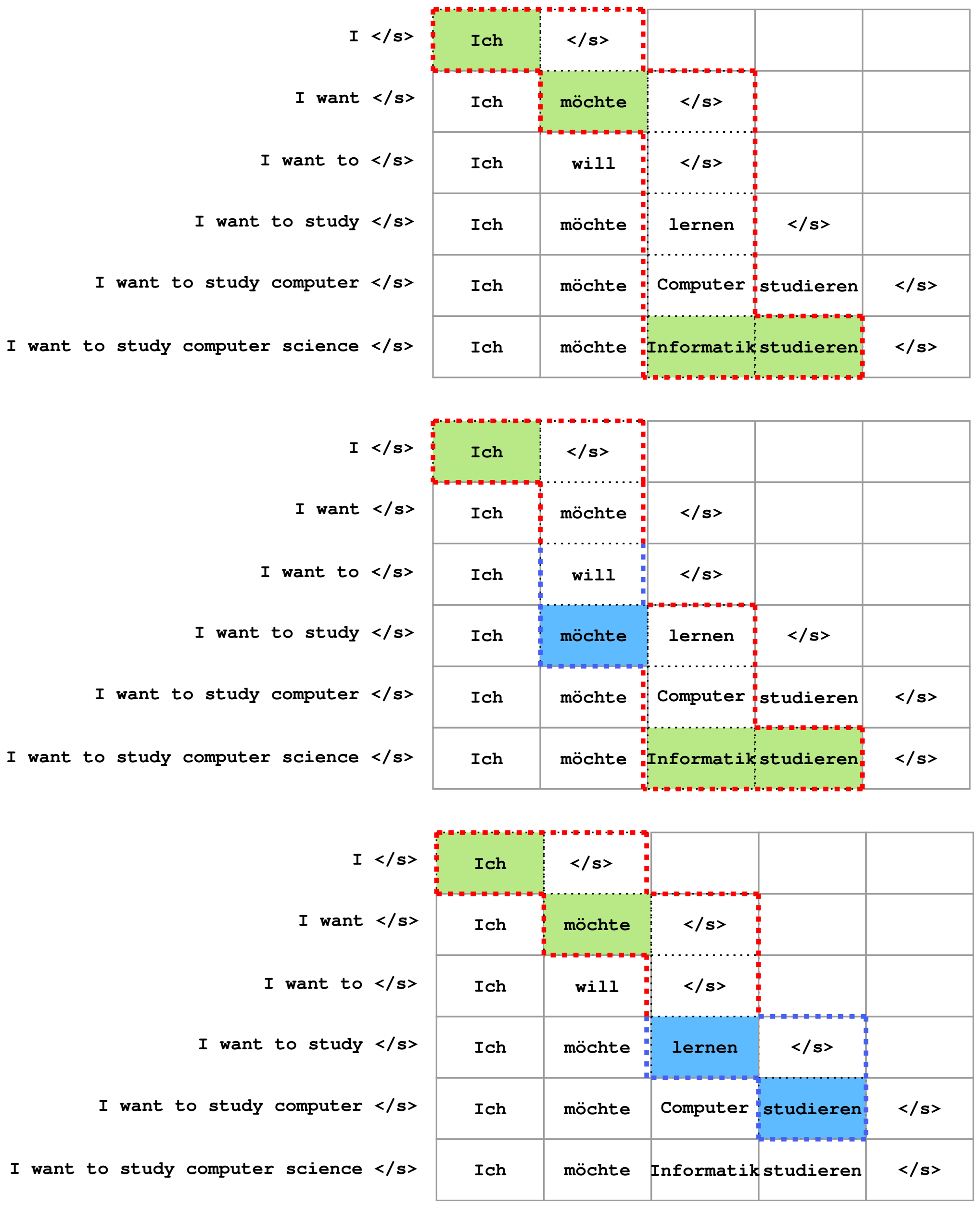
Harsh Satija and Joelle Pineau. 2016. Simultaneous machine translation using deep reinforcement learning. *Abstraction in Reinforcement Learning Workshop, ICML 2016*, (33).

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Hassan S Shavarani, Maryam Siahbani, Rantim M Seraj, and Anoop Sarkar. 2015. Learning segmentations that balance latency versus quality in spoken language translation. In *Proceedings of the Eleventh International Workshop on Spoken Language Translation (IWSLT 2015), Da Nang, Vietnam*, volume 26, page 52.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019a. Simpler and faster learning of adaptive policies for simultaneous translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1349–1354, Hong Kong, China. Association for Computational Linguistics.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019b. Simultaneous translation with flexible policy via restricted imitation learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5816–5822, Florence, Italy. Association for Computational Linguistics.



I </s>	Ich	</s>			
I want </s>	Ich	möchte	</s>		
I want to </s>	Ich	will	</s>		
I want to study </s>	Ich	möchte	lernen	</s>	
I want to study computer </s>	Ich	möchte	Computer	studieren	</s>
I want to study computer science </s>	Ich	möchte	Informatik	studieren	</s>

I </s>	Ich	</s>			
I want </s>	Ich	möchte	</s>		
I want to </s>	Ich	will	</s>		
I want to study </s>	Ich	möchte	lernen	</s>	
I want to study computer </s>	Ich	möchte	Computer	studieren	</s>
I want to study computer science </s>	Ich	möchte	Informatik	studieren	</s>

I </s>	Ich	</s>			
I want </s>	Ich	möchte	</s>		
I want to </s>	Ich	will	</s>		
I want to study </s>	Ich	möchte	lernen	</s>	
I want to study computer </s>	Ich	möchte	Computer	studieren	</s>
I want to study computer science </s>	Ich	möchte	Informatik	studieren	</s>

Figure 5: The first table shows an example of our oracle action sequence. In the middle table, we switch the second WRITE action to a READ action. We will continue reading until we observe the same word, then we can come back to the original path. In this scenario, the final translation does not change. In the last table, we distort the 5th READ into a WRITE action. Here, the generated translation is slightly changed as we are writing the wrong word.